МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«ВВЕДЕНИЕ В АРХИТЕКТУРУ ARM»

студента 2 курса, 23202 группы

Пятанова Михаила Юрьевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технических наук
Владислав Александрович
Перепёлкин

Новосибирск 2024

СОДЕРЖАНИЕ

ЦЕЛЬ

1. Знакомство с программной архитектурой ARM.
2. Анализ ассемблерного листинга программы для архитектуры ARM.

ОПИСАНИЕ РАБОТЫ

Полный компилируемый листинг реализованной программы и команда для её компиляции представлены в приложении 1.

Листинги на ассемблере с описаниями назначения команд с точки зрения реализации алгоритма выбранного варианта представлены в приложении 2 (уровни оптимизации O0-Og, не включая O3) и 3 (уровень оптимизации O3).

Была определена разница в ассемблерных листингах двух уровней оптимизации. См. приложение 3.

ЗАКЛЮЧЕНИЕ

Было проведено знакомство с программной архитектурой ARM. Проанализирован ассемблерный листинг программы для архитектуры ARM. Изучены основные отличия архитектур ARM и x86/x86-64: например, работа с регистрами.

Найдены отличия в ассемблерных листингах оптимизации уровней O0 и O3. Так в O3 отсутствую вызовы функций (кроме ввода, вывода, конвертации типов) – все функции были встроены (инлайнинг) в код, сложные операции были заменены простыми (умножение – сложением). Стоит отметить, что код был сильно разделен разными лейблами.

С субъективной точки зрения ARM ассемблер читается хуже x86/x86-64, после оптимизации стал читаться ещё хуже, однако понимание оптимизационных преобразований для платформы x86/x86-64 оказалось крайне полезным: компилятор проделал практически тоже самое.

Приложение 1.

Команды для компиляции:

Для получения ассемблерного листинга ARM GCC 14.2.0 использовался сайт: https://godbolt.org

```
#include <iostream>

long double GetSign(unsigned long long n) {
    return n % 2 == 0 ? 1.0 : -1.0;
}

long double GetPiNumberWithNAccuracy(unsigned long long N) {
    long double res = 0;
    for (unsigned long long i = 0; i <= N; i++) {
        res += (GetSign(i) / (2*i + 1));
    }
    return 4 * res;
}

int main() {
    unsigned long long n;
    scanf("%ull", &n);
    printf("%lf", GetPiNumberWithNAccuracy(n));
    return 0;
}
```

Приложение 2.

Ассемблерный листинг для архитектуры ARM с оптимизацией O0

1 _Z7GetSigny:	**Определяем функцию GetSign**
2 push {r7}	Пушим r7
3 sub sp, sp, #12	Резерв 12 байтов на стеке: sp = sp – 12
4 add r7, sp, #0	r7 = sp
5 strd r0, [r7]	Записываем из r0 r1 в память по адресу r7
6 ldrd r0, [r7]	Записываем из адреса в r7 в регистры r0 r1
7 and r2, r0, #1	Побитовое и r2 = r0 & 1
8 movs r3, #0	Записываем в r3 0
9 orrs r3, r3, r2	Побитовое или r3 = r3 | r2 с установкой Zero Flag
10 bne .L2	Если zero flag != 1, то L2
11 mov r2, #0	Зануляем r2 и r3
12 mov r3, #0	
13 movt r3, 16368	Заполняем младшие 16 бит 16368 n = 1;
14 b .L4	Переход в L4
15 .L2:	
16 mov r2, #0	Зануляем r2 и r3
17 mov r3, #0	
18 movt r3, 49136	Заполняем младшие 16 бит 49136 n = -1;
19 .L4:	
20 vmov d16, r2, r3	Записываем в d16 r2 r3
21 vmov.f64 d0, d16	Записываем знач. с плав. точкой из d16 в d0
22 adds r7, r7, #12	r7 = r7 + 12 восстанавливаем стек
23 mov sp, r7	sp = r7
24 ldr r7, [sp], #4	загружаем значение в r7 из памяти по адресу в sp и sp = sp + 4
25 bx lr	return
26 _Z24GetPiNumberWithNAccuracyy:	**Определяем функцию GetPiNumberWithNAccuracy**
27 push {r4, r5, r7, r8, r9, r10, fp, lr}	Записываем на стек регистры r4… и r4 будет сверху
28 vpush.64 {d8}	Пушим на стек значение с плав. точкой с регистра d8
29 sub sp, sp, #24	Вычитаем с указателя на верхушку стека 24(резерв)
30 add r7, sp, #0	Приравниваем r7 = sp
31 strd r0, [r7]	Записываем значения с r0 и r1 по адресу в r7
32 mov r2, #0	Обнуляем r2
33 mov r3, #0	Обнуляем r3
34 strd r2, [r7, #16]	Записываем с r2 и r3 в место по адресу r7+16 **long double res = 0**
35 vmov.i32 d16, #0 @ di	Записываем в NEON (SIMD&FP) регистр d16 значение 0 (@di dw int)
36 vstr.64 d16, [r7, #8] @ int	Сохраняем 64-разрядное значение из регистра NEON d16 в ячейке
37 b .L6	памяти на r7 + 8.(т.е. 0) **unsigned long long i = 0**
38 .L7:	Безусловный переход на L6
39 ldrd r0, [r7, #8]	Записываем в r0 r1 значение по ссылке с r7 + 8
40 bl _Z7GetSigny	Вызываем GetSign
41 vmov.f64 d8, d0	Записываем в d8 значение которое вернула функция
42 ldrd r2, [r7, #8]	Записываем в r2 r3 значение по адресу r7+8
43 adds r10, r2, r2	r10 = r2 + r2 = 2*r2 **(2*i)** с установкой carry flag
44 adc fp, r3, r3	r11 = r3 + r3 + carry flag
45 mov r2, r10	r2 = r10
46 mov r3, fp	r3 = r11
47 adds r8, r2, #1	r8 = r2 + 1
48 adc r9, r3, #0	r9 = r3 + carry flag
49 mov r0, r8	r0 = r8 (2i + 1)
50 mov r1, r9	r1 = r9
51 bl __aeabi_ul2d	вызываем функцию unsigned long long to a double-precision float
52 vmov d17, r0, r1	Записываем в d17 r0 r1
53 vdiv.f64 d16, d8, d17	Делим с плавающей точкой d16 = d8 / d17 **(GetSign(i) / (2*i + 1))**
54 vldr.64 d17, [r7, #16]	Записываем в d17 значение по адресу r7 + 16(**res**)
55 vadd.f64 d16, d17, d16	Складываем d16 = d17 + d16 **res += (GetSign(i) / (2*i + 1))**
56 vstr.64 d16, [r7, #16]	Сохраняем значение с d16 в r7 + 16
57 ldrd r2, [r7, #8]	Записываем в r2,r3 значение с r7 + 8
58 adds r4, r2, #1	Добавляем с флагом переноса r4 = r2 + 1
59 adc r5, r3, #0	r5 = r3 + 0 + carry
60 strd r4, [r7, #8]	Записываем в r7 + 8 значение с r4 **(i++)**
61 .L6:	
62 ldrd r2, [r7, #8]	Записываем в r2, r3 данные по ссылке из r7 + 8 (i)
63 ldrd r0, [r7]	Записываем в r0, r1 данные по ссылке из r7 (N)
64 cmp r0, r2	Сравниваем r0 и r2 (старшие биты)
65 sbcs r3, r1, r3	Вычитаем r3 = r1 – r3 и принимаем во внимание флаг переноса
66 bcs .L7	L7 только если флаг переноса поставлен **(i <= N)**
67 vldr.64 d16, [r7, #16]	
68 vmov.f64 d17, #4.0e+0	Записываем в d16 с r7 + 16
69 vmul.f64 d16, d16, d17	Записываем в d17 4
70 vmov.f64 d0, d16	Умножаем d16 = d16 * d17 = **res * 4**
71 adds r7, r7, #24	Записываем d16 в d0
72 mov sp, r7	r7 = r7 + 24 (прибираемся)
73 vldm sp!, {d8}	sp = r7

74 pop {r4, r5, r7, r8, r9, r10, fp, pc} 75 .LC0: 76 .ascii "%ull\000" 77 .LC1: 78 .ascii "%lf\000" 79 main: 80 push {r7, lr} 81 sub sp, sp, #8 82 add r7, sp, #0 83 mov r3, r7 84 mov r1, r3 85 movw r0, #:lower16:.LC0 86 movt r0, #:upper16:.LC0 87 bl __isoc23_scanf 88 ldrd r2, [r7] 89 mov r0, r2 90 mov r1, r3 91 bl _Z24GetPiNumberWithNAccuracyy 92 vmov r2, r3, d0 93 movw r0, #:lower16:.LC1 94 movt r0, #:upper16:.LC1 95 bl printf 96 movs r3, #0 97 mov r0, r3 98 adds r7, r7, #8 99 mov sp, r7 100 pop {r7, pc}	Записываем в d8 значение на которое указываем sp+8 и sp = sp +8 Восстанавливаем все регистры значениями со стека **Определяем функцию main** Кладем на стек в начале данные с lr(r14) потом с r7 Вычитаем из указателя на верхушку стека восемь(и присваиваем ему это значение. Записываем в r7 значение sp Записываем в r3 значение r7 Записываем в r1 значение r7 Записываем в r0 нижние 16 битов LC0 Записываем в r0 верхние 16 битов LC0 **Вызываем функцию scanf**. Регистр r0 используется для передачи параметров Записываем в r2, r3(как следующему) данные по ссылке из r7 Записываем в r0 данные с r2 Записываем в r1 данные с r3 **Вызываем функцию GetPiNumberWithNAccuracy** Записывает значение с плав. точкой с регистра d0 в r2(младшие 32 бита) и r3(старшие). Записываем в r0 нижние 16 битов LC1 Записываем в r0 верхние 16 битов LC1 **Вызываем функцию printf** **Return 0.** Записываем в регистр r3 значение 0 и обновляем флаг по значению 0. Записываем в r0 значение r3 Добавляем к указателю в r7 значение 8 Делаем указатель на стек равным r7 Удаляем со стека данные и записываем в r7 (восстановили данные лежавшие там до вызова программы), значение lr в pc. Таким образом отчистили память, которую использовали

Ассемблерный листинг для архитектуры ARM с оптимизацией O1

_Z7GetSigny:
tst r0, #1
bne .L3
vmov.f64 d0, #1.0e+0
bx lr
.L3:
vmov.f64 d0, #-1.0e+0
bx lr
_Z24GetPiNumberWithNAccuracyy:
push {r3, r4, r5, r6, r7, r8, r9, lr}
vpush.64 {d8, d9, d10, d11}
mov r9, r0
mov r8, r1
movs r4, #1
movs r6, #0
mov r5, r6
mov r7, r6
vmov.i64 d8, #0 @ float
vmov.f64 d11, #-1.0e+0
vmov.f64 d10, #1.0e+0
.L6:
and r3, r5, #1
cmp r3, #0
ite eq
vmoveq.f64 d9, d10
vmovne.f64 d9, d11
mov r0, r4
mov r1, r6
bl __aeabi_ul2d
vmov d17, r0, r1
vdiv.f64 d16, d9, d17
vadd.f64 d8, d8, d16
adds r2, r5, #1
adc r3, r7, #0
mov r5, r2
mov r7, r3
adds r4, r4, #2
adc r6, r6, #0
cmp r9, r2
sbcs r3, r8, r3
bcs .L6	

```
        vmov.f64        d0, #4.0e+0
        vmul.f64        d0, d8, d0
        vldm    sp!, {d8-d11}
        pop     {r3, r4, r5, r6, r7, r8, r9, pc}
.LC0:
        .ascii  "%ull\000"
.LC1:
        .ascii  "%lf\000"
main:
        push    {lr}
        sub     sp, sp, #12
        mov     r1, sp
        movw    r0, #:lower16:.LC0
        movt    r0, #:upper16:.LC0
        bl      __isoc23_scanf
        ldrd    r0, [sp]
        bl      _Z24GetPiNumberWithNAccuracyy
        vmov    r2, r3, d0
        movw    r0, #:lower16:.LC1
        movt    r0, #:upper16:.LC1
        bl      printf
        movs    r0, #0
        add     sp, sp, #12
        ldr     pc, [sp], #4
```

Ассемблерный листинг для архитектуры ARM с оптимизацией O2

```
_Z7GetSigny:
        lsls    r3, r0, #31
        bmi     .L3
        vmov.f64        d0, #1.0e+0
        bx      lr
.L3:
        vmov.f64        d0, #-1.0e+0
        bx      lr
_Z24GetPiNumberWithNAccuracyy:
        push    {r3, r4, r5, r6, r7, r8, r9, lr}
        movs    r6, #0
        mov     r9, r0
        vpush.64        {d8, d9}
        vmov.i64        d9, #0  @ float
        mov     r8, r1
        movs    r5, #1
        mov     r4, r6
        mov     r7, r6
.L7:
        vmov.f64        d8, #1.0e+0
        lsls    r3, r4, #31
        bpl     .L6
        vmov.f64        d8, #-1.0e+0
.L6:
        mov     r0, r5
        mov     r1, r6
        bl      __aeabi_ul2d
        vmov    d17, r0, r1
        adds    r4, r4, #1
        vdiv.f64        d16, d8, d17
        adc     r7, r7, #0
        adds    r5, r5, #2
        adc     r6, r6, #0
        cmp     r9, r4
        sbcs    r3, r8, r7
        vadd.f64        d9, d9, d16
        bcs     .L7
        vmov.f64        d0, #4.0e+0
        vmul.f64        d0, d9, d0
        vldm    sp!, {d8-d9}
        pop     {r3, r4, r5, r6, r7, r8, r9, pc}
.LC0:
        .ascii  "%ull\000"
.LC1:
        .ascii  "%lf\000"
main:
        push    {lr}
        movw    r0, #:lower16:.LC0
```

```
	movt	r0, #:upper16:.LC0
	sub	sp, sp, #12
	mov	r1, sp
	bl	__isoc23_scanf
	ldrd	r0, [sp]
	bl	_Z24GetPiNumberWithNAccuracyy
	vmov	r2, r3, d0
	movw	r0, #:lower16:.LC1
	movt	r0, #:upper16:.LC1
	bl	printf
	movs	r0, #0
	add	sp, sp, #12
	ldr	pc, [sp], #4
```

Ассемблерный листинг для архитектуры ARM с оптимизацией Os

```
_Z7GetSigny:
	lsls	r3, r0, #31
	bmi	.L3
	vmov.f64	d0, #1.0e+0
	bx	lr
.L3:
	vmov.f64	d0, #-1.0e+0
	bx	lr
_Z24GetPiNumberWithNAccuracyy:
	push	{r3, r4, r5, r6, r7, r8, r9, lr}
	movs	r6, #0
	mov	r9, r0
	vpush.64	{d8, d9}
	vmov.i64	d8, #0  @ float
	mov	r8, r1
	movs	r5, #1
	mov	r4, r6
	mov	r7, r6
.L6:
	lsls	r3, r4, #31
	bmi	.L7
	vmov.f64	d9, #1.0e+0
.L5:
	mov	r0, r5
	mov	r1, r6
	bl	__aeabi_ul2d
	vmov	d17, r0, r1
	adds	r4, r4, #1
	vdiv.f64	d16, d9, d17
	adc	r7, r7, #0
	adds	r5, r5, #2
	adc	r6, r6, #0
	cmp	r9, r4
	sbcs	r3, r8, r7
	vadd.f64	d8, d8, d16
	bcs	.L6
	vmov.f64	d0, #4.0e+0
	vmul.f64	d0, d8, d0
	vldm	sp!, {d8-d9}
	pop	{r3, r4, r5, r6, r7, r8, r9, pc}
.L7:
	vmov.f64	d9, #-1.0e+0
	b	.L5
.LC0:
	.ascii	"%ull\000"
.LC1:
	.ascii	"%lf\000"
main:
	push	{r0, r1, r2, lr}
	ldr	r0, .L10
	mov	r1, sp
	bl	__isoc23_scanf
	ldrd	r0, [sp]
	bl	_Z24GetPiNumberWithNAccuracyy
	vmov	r2, r3, d0
	ldr	r0, .L10+4
	bl	printf
	movs	r0, #0
	add	sp, sp, #12
	ldr	pc, [sp], #4
.L10:
	.word	.LC0
```

```
    .word   .LC1
```

Ассемблерный листинг для архитектуры ARM с оптимизацией Ofast

```
_Z7GetSigny:
        lsls    r3, r0, #31
        bmi     .L3
        vmov.f64        d0, #1.0e+0
        bx      lr
.L3:
        vmov.f64        d0, #-1.0e+0
        bx      lr
_Z24GetPiNumberWithNAccuracyy:
        push    {r3, r4, r5, r6, r7, r8, r9, lr}
        mov     r8, #0
        mov     r7, r0
        vpush.64        {d8, d9, d10}
        mov     r6, r1
        vmov.i64        d8, #0  @ float
        movs    r5, #1
        mov     r4, r8
        mov     r9, r8
        vmov.f64        d10, #1.0e+0
        vmov.f64        d9, #-1.0e+0
        b       .L9
.L13:
        bl      __aeabi_ul2d
        vmov    d17, r0, r1
        adds    r4, r4, #1
        vdiv.f64        d16, d9, d17
.L11:
        adc     r9, r9, #0
        adds    r5, r5, #2
        adc     r8, r8, #0
        cmp     r7, r4
        vadd.f64        d8, d8, d16
        sbcs    r3, r6, r9
        bcc     .L12
.L9:
        mov     r0, r5
        mov     r1, r8
        lsls    r3, r4, #31
        bmi     .L13
        bl      __aeabi_ul2d
        vmov    d17, r0, r1
        adds    r4, r4, #1
        vdiv.f64        d16, d10, d17
        b       .L11
.L12:
        vmov.f64        d0, #4.0e+0
        vmul.f64        d0, d8, d0
        vldm    sp!, {d8-d10}
        pop     {r3, r4, r5, r6, r7, r8, r9, pc}
.LC0:
        .ascii  "%ull\000"
.LC1:
        .ascii  "%lf\000"
main:
        push    {r4, r5, r6, r7, r8, r9, lr}
        movw    r0, #:lower16:.LC0
        movt    r0, #:upper16:.LC0
        vpush.64        {d8, d9, d10}
        sub     sp, sp, #12
        mov     r8, #0
        mov     r1, sp
        bl      __isoc23_scanf
        vmov.i64        d8, #0  @ float
        ldrd    r7, r6, [sp]
        movs    r5, #1
        mov     r4, r8
        mov     r9, r8
        vmov.f64        d10, #1.0e+0
        vmov.f64        d9, #-1.0e+0
        b       .L18
.L22:
        bl      __aeabi_ul2d
        vmov    d17, r0, r1
```

```
        adds    r4, r4, #1
        vdiv.f64        d16, d9, d17
.L20:
        adc     r9, r9, #0
        adds    r5, r5, #2
        adc     r8, r8, #0
        cmp     r7, r4
        vadd.f64        d8, d8, d16
        sbcs    r3, r6, r9
        bcc     .L21
.L18:
        mov     r0, r5
        mov     r1, r8
        lsls    r3, r4, #31
        bmi     .L22
        bl      __aeabi_ul2d
        vmov    d17, r0, r1
        adds    r4, r4, #1
        vdiv.f64        d16, d10, d17
        b       .L20
.L21:
        vmov.f64        d16, #4.0e+0
        movw    r0, #:lower16:.LC1
        movt    r0, #:upper16:.LC1
        vmul.f64        d16, d8, d16
        vmov    r2, r3, d16
        bl      printf
        movs    r0, #0
        add     sp, sp, #12
        vldm    sp!, {d8-d10}
        pop     {r4, r5, r6, r7, r8, r9, pc}
```

Ассемблерный листинг для архитектуры ARM с оптимизацией Og

```
_Z7GetSigny:
        tst     r0, #1
        bne     .L3
        vmov.f64        d0, #1.0e+0
        bx      lr
.L3:
        vmov.f64        d0, #-1.0e+0
        bx      lr
_Z24GetPiNumberWithNAccuracyy:
        push    {r3, r4, r5, r6, r7, lr}
        vpush.64        {d8, d9}
        mov     r7, r0
        mov     r6, r1
        movs    r4, #0
        mov     r5, r4
        vmov.i64        d9, #0  @ float
        b       .L5
.L6:
        mov     r0, r4
        mov     r1, r5
        bl      _Z7GetSigny
        vmov.f64        d8, d0
        adds    r0, r4, r4
        adc     r1, r5, r5
        adds    r0, r0, #1
        adc     r1, r1, #0
        bl      __aeabi_ul2d
        vmov    d17, r0, r1
        vdiv.f64        d16, d8, d17
        vadd.f64        d9, d9, d16
        adds    r4, r4, #1
        adc     r5, r5, #0
.L5:
        cmp     r7, r4
        sbcs    r3, r6, r5
        bcs     .L6
        vmov.f64        d0, #4.0e+0
        vmul.f64        d0, d9, d0
        vldm    sp!, {d8-d9}
        pop     {r3, r4, r5, r6, r7, pc}
.LC0:
        .ascii  "%ull\000"
.LC1:
```

```
        .ascii  "%lf\000"
main:
        push    {lr}
        sub     sp, sp, #12
        mov     r1, sp
        movw    r0, #:lower16:.LC0
        movt    r0, #:upper16:.LC0
        bl      __isoc23_scanf
        ldrd    r0, [sp]
        bl      _Z24GetPiNumberWithNAccuracyy
        vmov    r2, r3, d0
        movw    r0, #:lower16:.LC1
        movt    r0, #:upper16:.LC1
        bl      printf
        movs    r0, #0
        add     sp, sp, #12
        ldr     pc, [sp], #4
```

Приложение 3.

Ассемблерный листинг для архитектуры ARM с оптимизацией O3

1 _Z7GetSigny:	
2 lsls r3, r0, #31	**Компилятором был сделан инлайнинг, поэтому функции описаны ниже, внутри тела main.**
3 bmi .L3	
4 vmov.f64 d0, #1.0e+0	
5 bx lr	
6 .L3:	
7 vmov.f64 d0, #-1.0e+0	
8 bx lr	
9 _Z24GetPiNumberWithNAccuracyy:	
10 push {r3, r4, r5, r6, r7, r8, r9, lr}	
11 mov r8, #0	
12 mov r7, r0	
13 vpush.64 {d8, d9, d10}	
14 mov r6, r1	
15 vmov.i64 d8, #0 @ float	
16 movs r5, #1	
17 mov r4, r8	
18 mov r9, r8	
19 vmov.f64 d10, #1.0e+0	
20 vmov.f64 d9, #-1.0e+0	
21 b .L9	
22 .L13:	
23 bl __aeabi_ul2d	
24 vmov d17, r0, r1	
25 adds r4, r4, #1	
26 vdiv.f64 d16, d9, d17	
27 .L11:	
28 adc r9, r9, #0	
29 adds r5, r5, #2	
30 adc r8, r8, #0	
31 cmp r7, r4	
32 vadd.f64 d8, d8, d16	
33 sbcs r3, r6, r9	
34 bcc .L12	
35 .L9:	
36 mov r0, r5	
37 mov r1, r8	
38 lsls r3, r4, #31	
39 bmi .L13	
40 bl __aeabi_ul2d	
41 vmov d17, r0, r1	
42 adds r4, r4, #1	
43 vdiv.f64 d16, d10, d17	
44 b .L11	
45 .L12:	
46 vmov.f64 d0, #4.0e+0	
47 vmul.f64 d0, d8, d0	
48 vldm sp!, {d8-d10}	
49 pop {r3, r4, r5, r6, r7, r8, r9, pc}	
50 .LC0:	
51 .ascii "%ull\000"	
52 .LC1:	
53 .ascii "%lf\000"	
54 main:	**Определяем main**
55 push {r4, r5, r6, r7, r8, r9, lr}	Пушим на стек данные с r4…
56 movw r0, #:lower16:.LC0	Записываем в r0 нижние 16 битов LC0
57 movt r0, #:upper16:.LC0	Записываем в r0 верхние 16 битов LC0
58 vpush.64 {d8, d9, d10}	Пушим на стек SIMD&FP данные с регистров d8 d9 d10
59 sub sp, sp, #12	sp = sp – 12 (резерв 12 байт)
60 mov r8, #0	r8 = 0
61 mov r1, sp	r1 = sp
62 bl __isoc23_scanf	Вызов **scanf**
63 vmov.i64 d8, #0 @ float	d8 = 0.0 **res = 0**
64 ldrd r7, r6, [sp]	Записываем в r7(старшие) r6(младшие) биты с адреса sp **(N)**
65 movs r5, #1	r5 = 1 и обновляем флаг (Negative или Zero)
66 mov r4, r8	r4 = r8 **(i = 0)**
67 mov r9, r8	r9 = r8
68 vmov.f64 d10, #1.0e+0	d10 = 1 **результаты getSign**
69 vmov.f64 d9, #-1.0e+0	d9 = -1
70 b .L18	безусловный переход в L18
71 .L22:	
72 bl __aeabi_ul2d	Вызов перевода r0 (unsigned int) в r0 r1 (long double)

73 vmov d17, r0, r1	d17 = [r1 r0]
74 adds r4, r4, #1	r4++ **(i++)**
75 vdiv.f64 d16, d9, d17	d16 = d9 / d17 **(GetSign(i) = -1 / (2*i + 1)**
76 .L20:	
77 adc r9, r9, #0	r9 = r9 + carry flag **(i++)**
78 adds r5, r5, #2	r5 = r5 +2 set carry flag (второй счетчик заместо **2*i + 1**)
79 adc r8, r8, #0	r8 = r8 + carry flag
80 cmp r7, r4	сравниваем r7 r4
81 vadd.f64 d8, d8, d16	d8 = d8 + d16 **(res += (GetSign(i) / (2*i + 1))**
82 sbcs r3, r6, r9	r3 = r6 – r9 – carry flag и устанавливаем carry flag (i <= N) N - i
83 bcc .L21	L21 если carry flag = 0 т.е i > N
84 .L18:	r0 = r5
85 mov r0, r5	r1 = r8
86 mov r1, r8	r3 = r4 << 31 с установкой флага (Logical Shift Left Status)
87 lsls r3, r4, #31	L22 если установлен Negative Flag(идет проверка на четность)(нечет)
88 bmi .L22	Вызов перевода r0 (unsigned int) в r0 r1 (long double)
89 bl __aeabi_ul2d	d17 = [r1 r0]
90 vmov d17, r0, r1	r4++ с установкой флага
91 adds r4, r4, #1	d16 = d10 / d17 **(GetSign(i) = 1 / (2*i + 1)**
92 vdiv.f64 d16, d10, d17	безусловный переход L20
93 b .L20	
94 .L21:	d16 = 4
95 vmov.f64 d16, #4.0e+0	r0 = младшие LC1
96 movw r0, #:lower16:.LC1	r0 = старшие LC1
97 movt r0, #:upper16:.LC1	d16 = d8 * d16 **res*4**
98 vmul.f64 d16, d8, d16	[r3 r2] = d16
99 vmov r2, r3, d16	**Вызов printf**
100 bl printf	r0 = 0 с установкой флага **return 0**
101 movs r0, #0	sp = sp + 12
102 add sp, sp, #12	d8 = sp. Sp += 8. sp = d9. Sp += 8 …
103 vldm sp!, {d8-d10}	восстанавливаем прежние значения регистров (таким образом почистили память)
104 pop {r4, r5, r6, r7, r8, r9, pc}	